Error and Erasure Probabilities for Galileo Uplink Code

J. B. Berner
Telecommunications Systems Section

R. J. McEliece
California Institute of Technology

E. C. Posner
Office of Telecommunications and Data Acquisition

The Galileo uplink Frame Erasure probability and Undetected Frame Error probability are derived. The performance meets desired specifications under normal operations. The Galileo command system will work well even in an emergency condition, where the bit error rate into the command decoder is 1.00×10^{-3} (although Galileo's command threshold error rate is 1.00×10^{-5}).

I. Introduction

Galileo has a requirement for an undetected error rate for the End-to-End Command System of less than one undetected command error in 1×10^6 48-bit-long frames when the bit error rate into the CDU is 1×10^{-5} (Ref. 1). This translates into a bit error rate of 2.08×10^{-8}. While Galileo does not have any requirements on frame erasure rates, the draft Consultative Committee For Space Data Systems (CCSDS) command link coding recommendation is to have a 10^{-3} frame erasure probability rate and a 10^{-9} frame undetected error probability (Ref. 4).

These requirements are achieved with two forms of error detection/correction coding. All command frames are encoded with a block code, and messages spanning more than one frame are protected with an additional parity byte, called the *checksum byte*. If, due to the uplink coding, these requirements could be met at a higher input error probability, then the requirements on the signal-to-noise ratio (SNR) could be relaxed, which would allow more link margin. One purpose of this article is to analyze the effect that the uplink coding has on the channel error and erasure probabilities and to discover if any SNR savings can be found. A second purpose is to derive command error and erasure rates at the command threshold for use in making emergency commanding decisions.

II. The Code

The code that the Galileo uplink channel is using is a shortened cyclic code with the following generator polynomial (Ref. 1):

$$g(x) = x^7 + x^6 + x^2 + 1 = (x + 1)(x^6 + x + 1) \quad (1)$$

This generator polynomial gives a (63, 56) code. The version that Galileo uses is shortened to (47, 40) by filling the first sixteen bits of the longer code with 0's.

The second factor of the polynomial, $(x^6 + x + 1)$, is the generator polynomial for the (63, 57) Hamming code. The first term, $(x + 1)$, expurgates the code, i.e., it lessens the number of information bits by one and increases the number of parity bits by one. Thus, although the code can be thought of as a shortened BCH code (Ref. 1), the more useful way of describing it is as a shortened, expurgated Hamming code.

A by-product of expurgating the code is that all codewords are even weight. This fact will be used later in determining the code's error properties.

There are several ways of viewing the code. The two most useful both use the parity check matrix, H. The first way looks at the Hamming code parity check matrix and the second way looks at the cyclic properties of the unshortened code. Both ways will be used to demonstrate various properties and to obtain results.

It can be shown (Ref. 2) that the parity check matrix of the unshortened, expurgated code is:

$$H' = \begin{bmatrix} H \\ 1\ 1\ 1\ \ldots\ 1 \end{bmatrix} \tag{2}$$

where H is the parity check matrix of the (63, 57) Hamming code. Also, here the Hamming code is a linear code, of the form

$$H = [BI_{n-k}] \tag{3}$$

where I is the $(n - k) \times (n - k)$ identity matrix and B is a $(n - k) \times k$ matrix. Then, shortening H by t gives the following:

$$H'' = [B_t I_{n-k}] \tag{4}$$

where B_t is B minus the first t columns (Ref. 2). This result is easy to justify, since shortening a code is just replacing the first t information bits in the long code with 0's (and not sending these 0's), which zeros out the first t columns of the parity check matrix.

So the first goal is to get the shortened code into the form of Eq. (4). We do this by looking at the code in the second way, this time as a cyclic code.

The parity check matrix of a cyclic Hamming code can be written as:

$$H = [a^0, a^1, \ldots, a^{62}] \tag{5}$$

where the row vector [0100000] is equal to the primitive element a, which is a zero of the polynomial $x^6 + x + 1$ (Ref. 3). The Ith column of H is equal to $g(a^I)$. Since the unshortened code is cyclic, we can rotate the matrix and get:

$$H = [a^7, a^8, \ldots, a^{62}, a^0, \ldots, a^6] \tag{6}$$

This is now in the form of Eq. (3). So we shorten and get:

$$H'' = [a^{23}, \ldots, a^{62}, a^0, \ldots, a^6] \tag{7}$$

Thus, we can generate the necessary parity check matrix from the Galois field elements a^i. Note that the two parity check matrices H' and H'' are equivalent under row reduction operations.

III. Error Properties of the Code

Let us look at the code's parity check matrix H' (Eq. [2]) and analyze the error properties. The code will correct one error and detect two errors when used as an error correcting code; it will detect three errors when operating as an error detecting code.

The received code word, Y, is equal to the transmitted codeword, X, plus the error pattern, Z. The syndrome of the received codeword is equal to the parity check matrix, H', times the transpose of the received codeword. By definition of the parity check matrix, the syndrome of an unaltered codeword is the all 0 vector. Thus (Ref. 2),

$$\begin{aligned} \text{Syndrome} &= H' Y^T \\ &= H' (X^T + Z^T) \\ &= H' X^T + H' Z^T \\ &= H' Z^T \end{aligned}$$

So, we are only interested in the error patterns; the transmitted codeword does not affect the error properties. So from here on, for ease of calculation, we will assume that the all zero codeword is sent.

There are two distinct flavors of error patterns: odd weight and even weight, where weight is defined as the number of 1's in the codeword or error pattern. For more than two errors and the parity check matrix H', the following is true:

Even Weight Error Pattern:

$$H' Z^T = \begin{matrix} (a) \\ x \\ x \\ x \\ x \\ x \\ x \\ 0 \end{matrix} \quad \begin{matrix} (b) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

where x is either a 0 or a 1, Z^T is the transpose of the received codeword, and (a) and (b) are the only possible even weight error pattern syndromes.

The possible syndrome values for the Hamming code are the columns of the parity check matrix and the all 0 vector (the syndrome of a codeword) (Ref. 2). Since the parity check matrix H' has all 1's in its last row, the only even weight error patterns that will escape detection are the ones that are codewords. Thus, the code will detect error patterns with syndromes of pattern (a) and will not detect error patterns with syndromes of pattern (b).

Odd Weight Error Pattern:

$$H' Z^T = \begin{matrix} (c) \\ x \\ x \\ x \\ x \\ x \\ x \\ 1 \end{matrix} \quad \begin{matrix} (d) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{matrix}$$

If an odd weight error pattern occurs, the above syndromes are the only possible. The code will detect error patterns with syndromes of pattern (d), since the parity check matrix has no columns of this form, and will not detect error patterns with syndromes of pattern (c), unless the code is shortened. If the code is shortened, error patterns with syndromes equal to the zeroed out parity check matrix columns will be detected.

So, to determine the error statistics (how many detected and how many not detected for weight I), we must take a different approach for the even and odd weight errors. For the even weight errors, all we need is the weight enumerator (the number of words of weight I for all I) of the code; for the odd weight errors, we will have to do more work.

A. Even Weight Errors

There are 2^{40} possible code words; thus, counting the weights of the codewords borders on the impossible. However, using the dual code, the weight enumerator of the code can easily be found.

The dual code of a code is the code produced when using the parity check matrix as the generating matrix. This gives a code with $2^7 = 128$ codewords, easily countable with a computer. Then one can use the MacWilliams Identities (Ref. 2) to get the weight enumerator of the code from the weight enumerator of the dual code. The weight enumerator then gives the number of undetectable errors of weight I. Subtract the undetectable errors from the total number of errors of weight I, $\binom{47}{I}$, and we have the number of detectable even weight errors. The work is thus half done.

B. Odd Weight Errors

In the Galileo uplink, the code is used in two modes, error-detect (ED) and error-detect-and-correct (EDC). In error-detect mode, the code is used as an error detecting code; it does not try to correct the errors. This allows the code to detect all error patterns of weight three or less. In error-detect-and-correct mode, which is used only on data frames, the code is used as an error correcting code; thus, it can correct all one-error patterns and detect all two-error patterns.

In the error-detect mode, only even weight errors with the all 0 syndrome (patterns that are codewords) escape detection. Since the code has no odd weight codewords, if the code is operated in error detect mode, there are no undetectable odd weight errors.

However, if the code is operated in error-detect-and-correct mode, the situation is very different. The code will make an error if the error pattern causes a syndrome that is a column in the parity check matrix, i.e., the code assumes a single error has been made. When the code assumes a single error has been made, it will try to correct the error. The correction attempt changes one bit in the error pattern; this new bit may cancel out one error bit or it just adds one error bit. Thus, to generate an error pattern of weight $2K + 1$, there are $2K + 2$ ways of canceling a bit of a codeword of weight $2K + 2$ and there are $47 - 2K$ ways of adding a bit to a codeword of weight $2K$. So, the formula for the undetectable errors of weight $2K + 1$ is:

$$U = (47 - 2K) \times A_{2K} + (2K + 2) \times A_{2K+2} \qquad (8)$$

where A_i is the number of codewords of weight i.

Once again, the number of detectable errors is the number of error patterns of weight I minus the number of undetectable errors.

The weight enumerator, the number of detectable and undetectable errors, and the details of the calculations are given in Appendix A.

The next task is to compute for a given input (to the code) symbol error probability, p, the output frame error and erasure probabilities. This is straightforward:

For the error-detect-and-correct mode,

$$P_{\text{error}} = \sum_{i=2}^{47} p^i (1-p)^{47-i} U_i \tag{9}$$

$$P_{\text{erase}} = \sum_{i=2}^{47} p^i (1-p)^{47-i} D_i \tag{10}$$

where U_i and D_i are the error-detect-and-correct mode undetectable and detectable errors of weight i, respectively.

For the error-detect mode,

$$P_{\text{error}} = \sum_{i=1}^{47} p^i (1-p)^{47-i} U'_i \tag{11}$$

$$P_{\text{error}} = \sum_{i=1}^{47} p^i (1-p)^{47-i} D'_i \tag{12}$$

where U'_i and D'_i are the error-detect mode undetected and detected errors of weight i, respectively, as described above.

For an input symbol error of 1.00×10^{-5}, the output probabilities are as follows. For error-detect-and-correct,

$$P_{\text{error}} = 1.170 \times 10^{-11}$$

$$P_{\text{erase}} = 1.081 \times 10^{-7}$$

For error-detect,

$$P_{\text{error}} = 2.926 \times 10^{-17}$$

$$P_{\text{erase}} = 4.699 \times 10^{-4}$$

Figures 1 and 2 are plots of the error and erasure probabilities versus the input channel error probability for the error-detect-and-correct mode (Fig. 1) and the error-detect mode (Fig. 2). We see that the error-detect-and-correct mode provides error rates of 10^{-5} to 10^{-14} for $p = 10^{-3}$ to 10^{-6}, while the error-detect mode provides error rates of 10^{-9} to 10^{-21} for the same range of p.

IV. The Command Frame

A Galileo command frame is 48 bits long. It is equal to the 47-bit codeword, with a 0 fill bit appended to the end to make the frame length be a multiple of eight bits. The frame format (Ref. 1) is shown in Fig. 3.

There are two types of command frames, Single Frames and Multiple Frames (Ref. 1):

(1) Single Frame – A single command frame, decoded in error-detect mode.

(2) Multiple Frames – A header frame, called the *Block Command frame*, which is decoded in error-detect (ED) mode, followed by 1 to 31 Data frames, decoded in error-detect-and-correct (EDC) mode. The last information byte of the last data frame is the *checksum byte*, which is the result of exclusive-ORing all the previous information bytes. After decoding the data frames, the parity bits are discarded and the information bytes are exclusive-ORed. If the result is nonzero, an erasure is declared.

Thus, there will be two sets of error and erasure probabilities, one for the Single Frames and one for the Multiple Frames.

A. Single Frames

The Single Frame is just a codeword decoded in the error-detect mode. The resulting error and erasure probabilities are shown in Fig. 2. For an input symbol error probability of 1.00×10^{-5}, the Single Frame probabilities are:

$$P_{\text{error}} = 2.926 \times 10^{-17}$$

$$P_{\text{erase}} = 4.699 \times 10^{-4}$$

From Fig. 2, we see that the coding provides error rates that are at least 10^{-8} below the erasure rates.

B. Multiple Frames

Before finding the required probabilities, we must understand how the overall decoding system (Hamming Code decoder and *checksum byte*) works.

The *Block Command frame* is decoded. If errors are detected, processing of the Multiple Frame stops and an erasure

is declared. If no errors are detected, the Data frames are decoded. If an uncorrectable error is detected, processing of the Multiple Frame stops and an erasure is declared. If all frames are decoded without an erasure, the data bytes (the 40 information bits per Data frame) are exclusive-ORed, and if the result is nonzero, the processing is stopped and an erasure is declared. The last data byte of the last Data frame is called the *checksum byte*, since this byte is appended onto the data stream to make the exclusive-ORing of the data bytes be zero (Ref. 1).

Since the code and checksum are linear, any frame can be used for the analysis. So we will assume that the all zero frame is sent; thus, a 1 denotes an error.

Assume that the codeword has passed through the decoder. We now drop the 7 parity bits and only consider the 40 information bits (five information bytes). If we break each byte into its 8 bits and look at each bit position (or slot) in the byte, two things are obvious:

(1) If there is an odd number of 1's in any slot, the exclusive-ORing process will result in a 1 being in that slot; thus, an erasure will be declared.

(2) If there is an even number of 1's in each slot, the exclusive-ORing process will result in a 0 being in that slot; so, no error will be detected.

Thus, for an error pattern to escape detection, there must be an even number of 1's in each slot.

Now, the statistics of the shortened, expurgated Hamming code are such that if the decoder makes an error, it outputs even weight error patterns of weight four or more. And, as was shown above, if there is an odd number of channel errors that the decoder cannot correct or detect, the decoder outputs either one less error or one more error. Thus, the minimum number of channel errors at the input to the decoder to cause an error that is undetectable (or uncorrectable) by the decoder is three. So the probability of an error that is undetectable by the checksum is proportional to p^3, for small p.

Since we have multiple data frames, the question is, do we worry about multiple frame errors, or do we just look at a single frame in error? The probability of a multiple undetected frame error would be equal to $C \times p^6 + O(7)$ (since p^3 is the minimum for a single frame), where C is a constant. Since p is less than 10^{-3}, the probability of a double frame error is, at the least, 10^{-9} less than the probability of a single frame error. Thus, we can ignore all multiple data frame errors and concentrate on undetected single data frame errors. And, since we are dealing with a low p, the number of undetectable combinations we have to look at is small.

The undetectable error patterns are those with an even number of 1's in each slot. This only occurs in error patterns with an even number of 1's in the information bits.

There are 80 possible undetectable weight 2 information-bit error patterns. The corresponding codeword weights are (see Appendix B for the derivation):

37	weight 4
32	weight 6
11	weight 8

There are 2840 possible undetectable weight 4 information-bit error patterns. The corresponding codeword weights are:

37	weight 4
892	weight 6
1591	weight 8
320	weight 10

There are 58800 possible undetectable weight 6 information bit error patterns. The corresponding codeword weights are:

924	weight 6
19253	weight 8
32244	weight 10
6379	weight 12

Before we begin, some symbols that we will use:

P_R^{ED} = Probability that the decoder declares an erasure in ED mode

P_R^{EDC} = Probability that the decoder declares an erasure in EDC mode.

P_E^{ED} = Probability that the decoder makes an error in ED mode

P_E^{EDC} = Probability that the decoder makes an error in EDC mode

With the above numbers in hand, and with the knowledge that the code converts odd weight error patterns into even weight patterns, we can calculate the probability that one frame will have an error pattern that is undetectable by the *checksum* (called P_u), given that the decoder made an error.

$$\left.\begin{aligned} P_u &= \left(\begin{array}{l} \text{sum of (the number of patterns of weight} \\ i \text{ that are undetectable)} \end{array} \right. \\ &\quad \left. \times \frac{p^i \times (1-p)^{n-i}}{P_E^{EDC}} \right) \\ P_u &= \frac{(45 \times 0 + 4 \times (37+37)) \times p^3 \times (1-p)^{44}}{P_E^{EDC}} \\ &\quad + \frac{(37+37) \times p^4 \times (1-p)^{43}}{P_E^{EDC}} \\ &\quad + \frac{(43 \times (37+37) + 6 \times (32+892+924))}{P_E^{EDC}} \\ &\quad \times p^5 \times (1-p)^{42} \\ &\quad + \frac{0\,(p^6)}{P_E^{EDC}} \end{aligned}\right\} \quad (13)$$

$$P_u = \frac{(296p^3 (1-p)^{44} + 74p^4 (1-p)^{43} + 14270p^5 (1-p)^{42})}{P_E^{EDC}} \quad (14)$$

And the probability of a frame having a detectable error by the checksum (P_d), given that the decoder made an error, is:

$$P_d = 1 - P_u \quad (15)$$

And we will assume that we have n data frames, where n is between 1 and 31.

Now let's calculate the erasure probability of the system, remembering that an erasure stops the processing:

$$\left.\begin{aligned} P_{\text{erase}} &= \text{Prob (erasure in Block Command frame)} \\ &\quad + \text{Prob (erasure in one of the } n \text{ data frames)} \\ &\quad + \text{Prob (checksum declares erasure)} \\ P_{\text{erase}} &= P_R^{ED} + (1 - P_R^{ED}) \times P_R^{EDC} \\ &\quad \times \sum_{i=1}^{n-1} (1 - P_R^{EDC})^{i-1} \\ &\quad + \binom{n}{1} \times P_d \times (1 - P_R^{EDC} - P_E^{EDC})^{n-1} \\ &\quad \times (1 - P_R^{ED}) \times P_E^{EDC} \end{aligned}\right\} \quad (16)$$

And, the error probability (again assuming one undetectable error):

$$\left.\begin{aligned} P_{\text{error}} &= \text{Prob (error in Block Command frame)} \\ &\quad + \text{Prob (undetectable error in data frame)} \\ P_{\text{error}} &= P_E^{ED} \times (1 - P_E^{EDC} - P_R^{EDC})^n \\ &\quad + (1 - P_E^{ED} - P_R^{ED}) \times \binom{n}{1} \times P_u \\ &\quad \times (1 - P_E^{EDC} - P_R^{EDC})^{n-1} \times P_E^{EDC} \end{aligned}\right\} \quad (17)$$

The resulting probabilities are plotted in Figs. 4, 5, and 6, for n = 5, 10, and 31. For a channel error probability of $p = 10^{-5}$, the results are:

For $n = 5$,

$$P_{\text{error}} = 1.48 \times 10^{-12}$$

$$P_{\text{erase}} = 4.70 \times 10^{-4}$$

For $n = 10$,

$$P_{\text{error}} = 2.96 \times 10^{-12}$$

$$P_{\text{erase}} = 4.71 \times 10^{-4}$$

For $n = 31$,

$$P_{\text{error}} = 9.17 \times 10^{-12}$$

$$P_{\text{erase}} = 4.73 \times 10^{-4}$$

V. Examining the Results

Let us assume the CCSDS requirements (10^{-3} frame erasure probability and 10^{-9} frame undetected error probability). At least for now, we will assume these requirements for the Multiple Frame. So, using Figs. 2, 4, and 5, we get the following undetected error probabilities at an erasure probability of 10^{-3}:

For $n = 5$,

$$P_{\text{error}} = 1.00 \times 10^{-11}$$

For $n = 10$,

$$P_{\text{error}} = 2.50 \times 10^{-11}$$

For $n = 31$,

$$P_{\text{error}} = 8.00 \times 10^{-11}$$

These values occur at a channel error probability of 2.00×10^{-5} or $1.00 \times 10^{-4.70}$, which gives an E_b/N_0 improvement of 0.33 dB over the E_b/N_0 required for a 1.00×10^{-5} channel error probability.

Now, let us consider the $n = 31$ case (this has the highest error probability). The error probability is equal to $1.00 \times 10^{-10.09691}$. This means we have one frame in error for every $10^{10.09691}$ frames. At 32 bits per second, and 48 bits per frame, that means that it will take 594 years for an error to occur. Or, to put it another way, we could have started transmitting commands continuously when Galileo was alive in the mid 1600's and still would not expect an error until sometime in the twenty-third century (Jim Taylor, private communication).

VI. Command Threshold

We now want to look at what happens at the command threshold. The command threshold is assumed to be the point where the input symbol error rate, p, is 1.00×10^{-3}.

In Section IV, we made an assumption that since the symbol error rate would be less than 10^{-3}, we only had to worry about one frame in error when we have a multiple frame command. Since we are at $p = 10^{-3}$, we should look at this again:

For the probability of erasure, P_{erase}, the only term affected is Prob (*checksum* declares erasure). This term is equal to the sum of Prob (*checksum* declares erasure | number of frame errors = i) $\times$ Prob (number of frame errors = i). For each i, the summed terms are bounded by:

$$\binom{n}{i} \times (P_{\text{E}}^{\text{EDC}})^i$$

At $p = 10^{-3}$, $P_{\text{E}}^{\text{EDC}} = 1.1 \times 10^{-5}$. The later terms in the summation are thus insignificant when compared to the first term (i equals 1). Thus the equation for P_{erase} (Eq. [16]) holds at $p = 10^{-3}$.

The probability of error is even easier. The probability, P_u, of an undetectable error, given that a frame error has occurred, is the summation over i of $D \times (p^3)^i$, where i is the number of frames that have errors and D is a constant that depends on i. With $p = 10^{-3}$, only the first term (i equals 1) is significant. So the equation for P_{error} (Eq. [17]) holds.

The equations give the following numbers for $p = 10^{-3}$:

Single Frame:

$$P_{\text{erase}} = 4.6 \times 10^{-2}$$

$$P_{\text{error}} = 2.8 \times 10^{-9}$$

Multiple Frame:

$$n = 5, \quad P_{\text{erase}} = 5.0 \times 10^{-2}$$

$$P_{\text{error}} = 1.3 \times 10^{-6}$$

$$n = 10, \quad P_{\text{erase}} = 5.5 \times 10^{-2}$$

$$P_{\text{error}} = 2.7 \times 10^{-6}$$

$$n = 31, \quad P_{\text{erase}} = 7.6 \times 10^{-2}$$

$$P_{\text{error}} = 8.1 \times 10^{-6}$$

So, even in the worst case ($n = 31$), in an emergency, only 7.6% of the frames would get erased and more importantly, those that get through the decoder have an error probability of 8.1×10^{-6}. This means that it is safe to blind command (the sending of commands without any hope of the spacecraft verifying them) Galileo in case of a spacecraft emergency.

VII. Summary

The Galileo uplink frame erasure and undetectable frame error probabilities have been derived. It has also been shown that the Expurgated Hamming code with *checksum byte* gives better than required error protection for a given erasure probability.

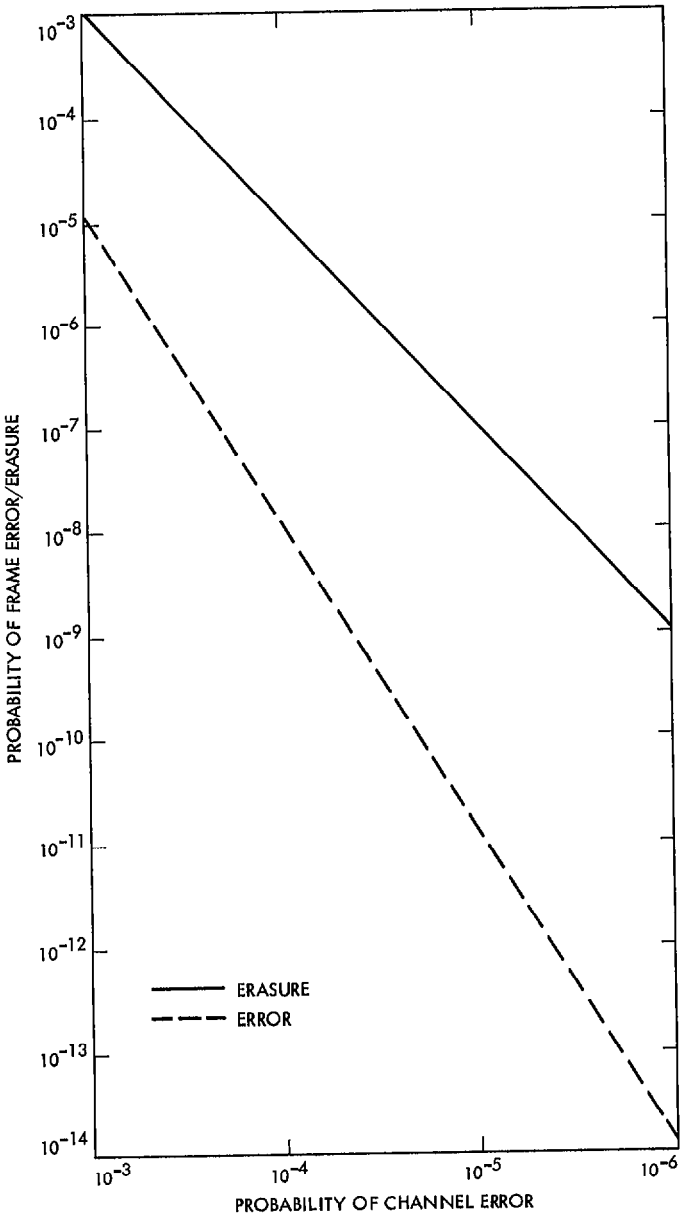

Fig. 1. Probabilities for error correct mode

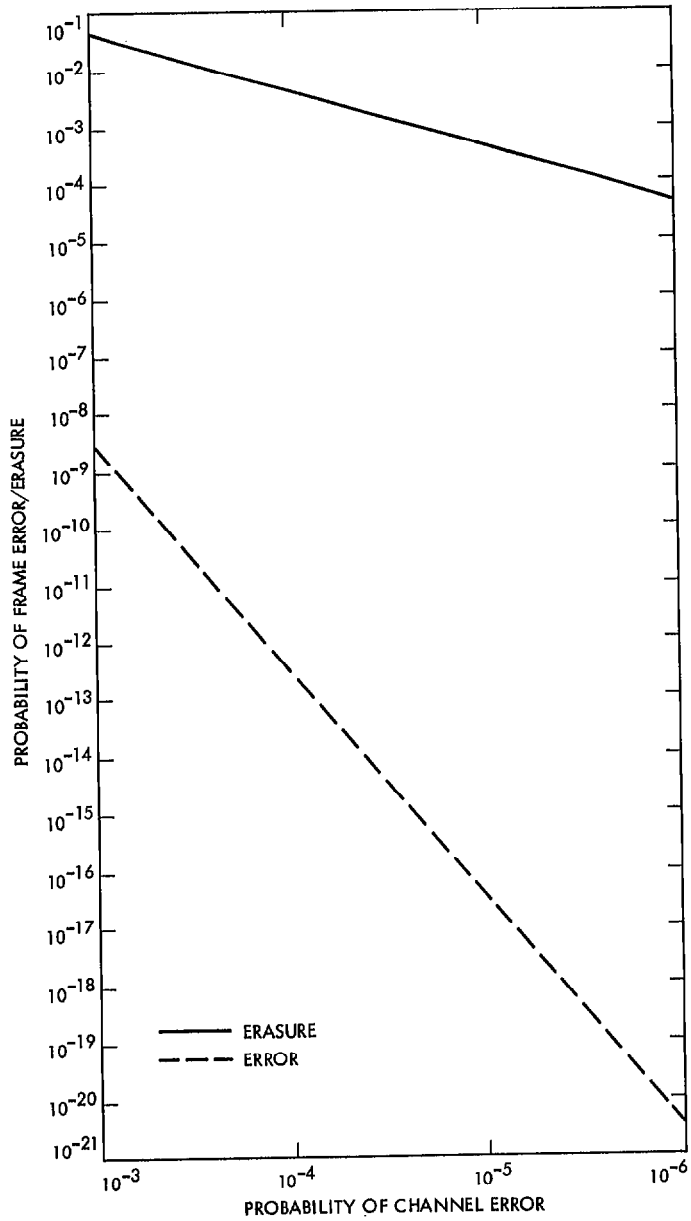

Fig. 2. Probabilities for error detect mode

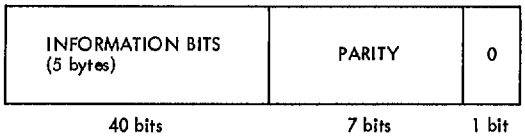

Fig. 3. Galileo command frame format

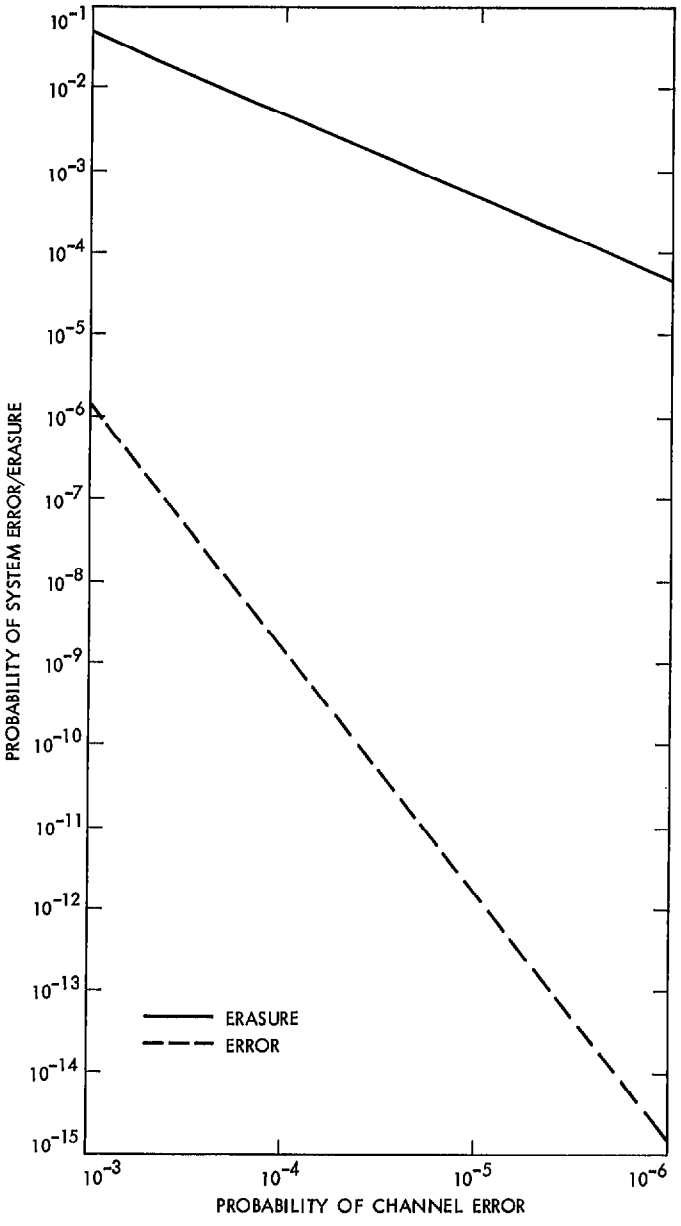

Fig. 4. Probabilities for multiple frames where $n = 5$

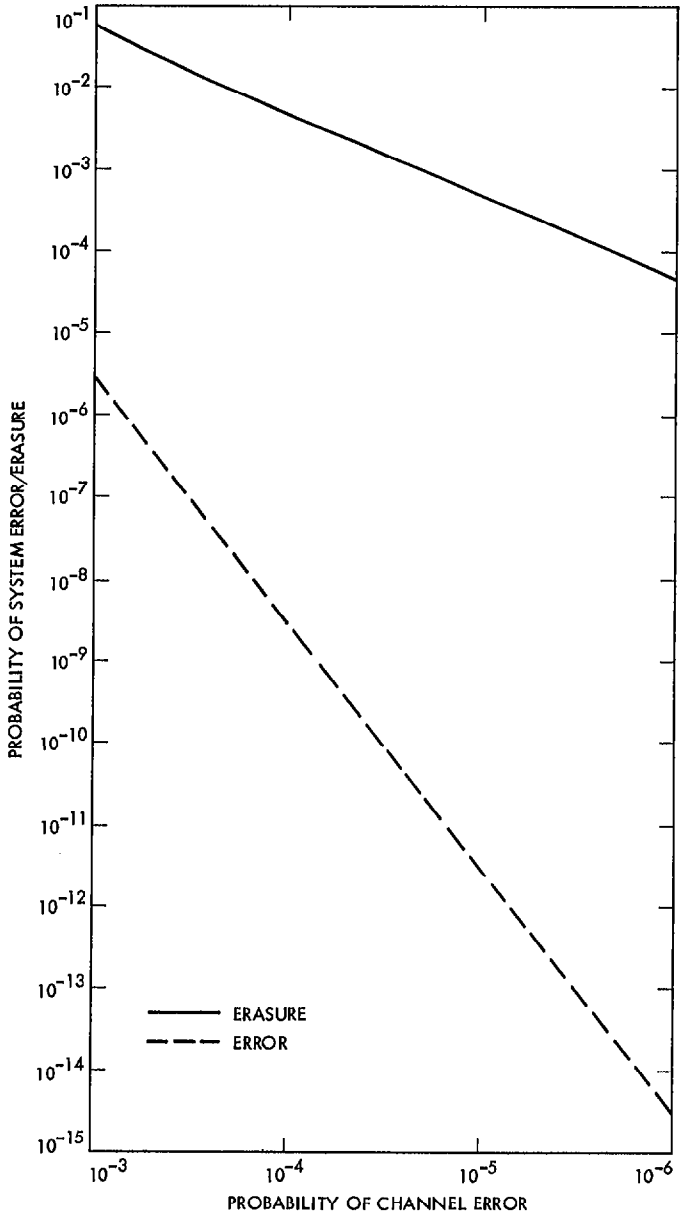

Fig. 5. Probabilities for multiple frames where $n = 10$

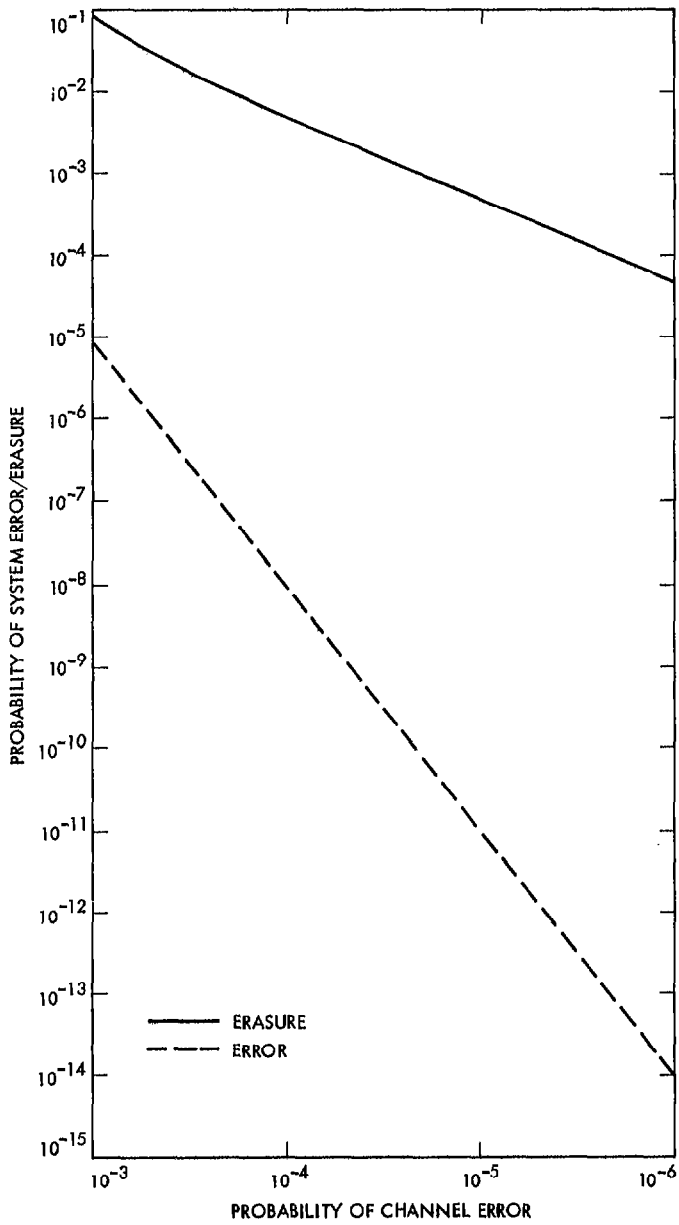

Fig. 6. Probabilities for multiple frames where $n = 31$

Appendix A
Code Statistics

The goal is to find, for each error pattern weight, the number of detectable and undetectable error patterns. To do this, we must first find the weight enumerator of the code.

The weight enumerator is found via the dual code of the code. The dual code is the code created by using the parity check matrix of our code as a generator matrix. This creates a (47, 7) code, which has only $2^7 = 128$ codewords. The weight enumerator of this code is easily found by computer by generating the 128 7-bit sequences, encoding them with the parity check matrix, and then counting the 1's in each codeword. The results of this are in Table A-1.

The next step is to use the MacWilliams identities to calculate the weight enumerator of the code. The MacWilliams identities state (Ref. 2) that if $B(z)$ is the weight enumerator of an (n, k) linear code and $A(z)$ is the weight enumerator of its dual code, then:

$$A(z) = 2^{-k} \sum_{i=0}^{n} B_i (1-z)^i (1+z)^{n-i} \qquad \text{(A-1)}$$

In our case, $k = 7$, $n = 47$, $B(z)$ is the weight enumerator of the dual code, and $A(z)$ is the weight enumerator of the code. So, again using a computer, the weight enumerator can be calculated. The results are shown in Table A-2. Now we can calculate the number of detectable and undetectable error patterns of weight i. First, let's do it for the EDC mode.

As was discussed in Section III, the only undetectable even weight error patterns are those that are codewords. Thus, for errors of weight $2i$, the number of undetectable and detectable patterns are:

$$\text{Undetectable} = A(2i) \qquad \text{(A-2)}$$

$$\text{Detectable} = \binom{47}{2i} - A(2i) \qquad \text{(A-3)}$$

And, as was shown in Section III, the number of odd weight detectable and undetectable error patterns is:

$$\text{Undetectable} = (47 - 2i) \times A(2i) + (2i + 2) \times A(2i + 2) \qquad \text{(A-4)}$$

$$\text{Detectable} = \binom{47}{2i+1} - (47 - 2i) \times A(2i) - (2i + 2) \times A(2i + 2) \qquad \text{(A-5)}$$

The results are shown in Table A-3.

For the ED mode, the even weight error pattern breakdown is the same. The difference is in the odd weight error patterns. Since we are in ED mode, errors that look like single errors are detected, not corrected. So, there are no undetectable error patterns of odd weight. These results are in Table A-4.

Table A-1. Dual code weight enumerator

Weight	Number of Words
0	1
19	3
20	4
21	5
22	23
23	28
24	28
25	23
26	5
27	4
28	3
47	1

Table A-2. Weight enumerator

Weight	Number of Words
0	1
2	0
4	2927
6	167017
8	4916447
10	80897478
12	816454377
14	5338125069
16	23488120074
18	71384955784
20	152538959670
22	231779439554
24	251934559006
26	196121173412
28	108956243986
30	42831113394
32	11743967013
34	2198099208
36	272132459
38	21294613
40	981971
42	24070
44	245
46	1

Table A-3. Error statistics (EDC mode)

Weight	Detectable	Undetectable
1	47	0
2	1081	0
3	4507	11708
4	175438	2927
5	405976	1127963
6	10570556	167017
7	16712226	46179273
8	309541048	4916447
9	361932932	1000716213
10	5097169273	80897478
11	4626474407	12790659210
12	51434946474	816454377
13	37367194284	103309654161
14	336305649726	5338125069
15	199648256088	551968048461
16	1479744489024	23488120074
17	728127949008	2013060926406
18	4497263169906	71384955784
19	1852256859654	5120942911136
20	9609940719436	152538959670
21	3334060006144	9217699581278
22	14602118254672	231779439554
23	4282886436556	11840915404994
24	15871867282544	251934559006
25	3940252328376	10893645365850
26	12355638414010	196121173412
27	2593160205846	7169319473260
28	6864243526804	108956243986
29	1213546088136	3355102037554
30	2698357762020	42831113394
31	399296736984	1103935872114
32	739872337536	11743967013
33	90748896528	250894878267
34	138478749237	2198099208
35	13879342623	38372058228
36	17145001158	272132459
37	1375414408	3802652343
38	1341354532	21294613
39	83527138	230930357
40	61909528	981971
41	2852836	7884737
42	1509869	24070
43	47235	131130
44	15970	245
45	300	781
46	46	1
47	0	1

Table A-4. Error statistics (ED mode)

Weight	Detectable	Undetectable
1	47	0
2	1081	0
3	16215	0
4	175438	2927
5	1533939	0
6	10570556	167017
7	62891499	0
8	309541048	4916447
9	1362649145	0
10	5097169273	80897478
11	17417133617	0
12	51434946474	816454377
13	140676848445	0
14	336305649726	5338125069
15	751616304549	0
16	1479744489024	23488120074
17	2741188875414	0
18	4497263169906	71384955784
19	6973199770790	0
20	9609940719436	152538959670
21	12551759587422	0
22	14602118254672	231779439554
23	16123801841550	0
24	15871867282544	251934559006
25	14833897694226	0
26	12355638414010	196121173412
27	9762479679106	0
28	6864243526804	108956243986
29	4568648125690	0
30	2698357762020	42831113394
31	1503232609098	0
32	739872337536	11743967013
33	341643774795	0
34	138478749237	2198099208
35	52251400851	0
36	17145001158	272132459
37	5178066751	0
38	1341354532	21294613
39	314457495	0
40	61909528	981971
41	10737573	0
42	1509869	24070
43	178365	0
44	15970	245
45	1081	0
46	46	1
47	1	0

Appendix B

Undetectable Information-Bit Error Patterns

The only way for errors to escape detection by the *checksum byte* is to have an even number of errors in each bit slot. Since there are 5 information bytes per Data frame, there are 5 different bits per bit slot. We are only interested in the undetectable patterns with 2, 4, and 6 information bits in error.

I. Two Information Bit Errors

The only way to get an undetectable error pattern with two information bits in error is to have both bits in the same slot. That means that there are $\binom{5}{2}$ ways to do this in each slot and that there are $\binom{8}{1}$ slots to do it in. So:

$$\binom{5}{2} \times \binom{8}{1} = 80$$

There are 80 possible two information-bit error patterns that will escape detection. To find the codeword weights, a computer was used; each pattern was generated, encoded, and then "weighed" (the number of 1's counted). The results are:

37	weight 4
32	weight 6
11	weight 8

II. Four Information Bit Errors

There are two types of four information-bit error patterns that escape detection. The first is all four errors in one slot. There are $\binom{5}{4}$ ways per slot to do this and $\binom{8}{1}$ slots to do it in. Secondly, there could be two sets of slots with two errors in each. There are $\binom{5}{2}$ ways per slot and $\binom{8}{2}$ slots to do it in. Thus:

$$\binom{8}{1} \times \binom{5}{4} + \binom{8}{2} \times \binom{5}{2} \times \binom{5}{2} = 2840$$

So, there are 2840 possible four information bit patterns. Again a computer was used to generate the codeword weights, using the same method as before. The results are:

37	weight 4
892	weight 6
1591	weight 8
320	weight 10

III. Six Information Bit Errors

There are two types of six information-bit error patterns that escape detection. The first has three slots with two errors in each slot. There are $\binom{5}{2}$ ways per slot and $\binom{8}{3}$ slots to do it in. The second way has four errors in one slot and two errors in another. There are $\binom{5}{4}$ ways to do it in the four error slot and $\binom{5}{2}$ ways to do it in the two error slot; there are 8×7 slots to do it in. So:

$$8 \times 7 \times \binom{5}{4} \times \binom{5}{2} + \binom{8}{3} \times \binom{5}{2} \times \binom{5}{2} \times \binom{5}{2} = 58800$$

Thus, there are 58800 undetectable six information-bit error patterns. Using the same methods as above, the codeword weights were calculated. The results are:

924	weight 6
19253	weight 8
32244	weight 10
6379	weight 12

References

1. *Project Galileo Orbiter Functional Requirements Book, Vol. I*, Document GLL-3-290, Rev. B, "Command Structure and Assignment" (internal document), Jet Propulsion Laboratory, Pasadena, California.

2. McEliece, Robert J., *The Theory of Information and Coding*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1977.

3. Blahut, Richard E., *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

4. *Telecommand, Part-1: Channel Service*, CCSDS Draft Recommendation, Red Book, Issue-1, Consultative Committee for Space Data Systems, Washington, D.C., April 1985.